

Bachelor's Degree in Audiovisual System
Engineering

Academic course 2017/2018

Bachelor Thesis

Music composition based on
Artificial Neural Networks

A PROOF OF CONCEPT ON MELODY IMPROVISATION

Clara Luis Mingueza

Tutor

Carmen Peláez Moreno

19th June 2018

uc3m | Universidad **Carlos III** de Madrid



This work is licensed under Creative Commons
Attribution – Non Commercial – Non Derivatives

Acknowledgment

I would like to thank my parents for always letting me go ahead with my dreams. That is, by far, what makes me the happiest.

Thanks to Axel, my constant and support.

Thanks to Nerea, my sister, best friend and confident.

Thanks to Carmen, for always being the guiding light that leads the project and for trusting in me even at the hardest moments. I feel very grateful for having you as a tutor.

Thanks to the T3chFest family and to the Student Union for being a shelter whenever I needed it.

Thanks to Ruth, María, Patri and Carmen for making the day-to-day in the career a little happier.

Thanks, sincerely, to all the people that completed the online listening test. It is daunting to receive so many responses, social media support and even wishes for knowing the results. It has been amazing to know that a hard task as music generation has had some good results.

Abstract

In the recent years, research on Artificial Intelligence has ushered in a new phase of technology evolution. Autonomous systems such as voice assistants or self-driving cars are a present reality as first commercial systems have been already launched to the market.

New applications emerge each year as huge amounts of generated data and computational capabilities make the development of accurate and expert systems plausible. This evolution is optimizing processes of many core fields such as agriculture, telecommunications or medicine.

A quite technological field such as music is also beginning to notice changes, as recommendation engines, synthesizers and music generation are attractive fields of research with some preliminary results.

With this project, we intend to contribute to ease the process of music creation making it more accessible to people. The subject of this project is the design, development and experimentation of an AI engine to generate music. A simple, but pleasant to hear artificially generated melody could serve as a base for people to compose more complex pieces of music.

At the same time, the project sheds some light to the nuts and bolts of novel techniques for music composition, as the Long Short Term Memory network selected.

The system processes MIDI files and extracts relevant information for training the network. The extracted data has been selected by analyzing the main aspects used in the field of Music Information Retrieval.

An online listening test taken by subjects of different musical backgrounds is designed to measure the quality of the artificial composer. The final results prove that pleasant to hear melodies have been composed.

Keywords Artificial intelligence; Autonomous systems; Supervised learning; Machine learning; Artificial neural networks; Recurrent neural networks; Computer generated music; Long Short Term Memory

Contents

List of Figures	8
List of Tables	10
1 Introduction	12
1.1 Motivation	13
1.2 Objectives	14
1.3 Regulatory Framework	14
1.3.1 Legal analysis in Artificial Intelligence	14
1.3.2 MIDI as a technical standard	15
1.4 Socio-economic Environment	16
1.4.1 Budget	16
1.4.2 Socio-economic impact	18
2 State of the Art	19
2.1 MIR Overview	19
2.2 A brief on music theory	20
2.2.1 Pitch: Notes	20
2.2.2 Harmony: Scales and Intervals	21
2.2.3 Temporal: Notes, rests and tempo	23
2.3 Computational music description	24
2.3.1 Music N (1957 - 1986) and CSound	25
2.3.2 MIDI	26
2.4 The role of machine learning for music generation	28
2.4.1 Artificial Neural Networks	31
2.4.2 Deep Learning	33
2.4.3 Recurrent Neural Networks	35
2.4.4 Long Short Term Memory networks (LSTMs)	36
2.5 Related works	37
2.5.1 Magenta project	37
2.5.2 Blues improvisation	38
2.5.3 Konstantin Lackner's Composer	39
3 System design	40
3.1 Initial approach	40

3.2	The final chosen system	41
3.2.1	Programming Language	42
3.2.2	Frameworks	43
3.2.3	Dataset	45
3.2.4	Network architecture	47
4	Experiments and Results	49
4.1	Preliminary experiments	49
4.1.1	General observations	50
4.2	Final music generations	51
4.3	Online Listening Test	52
4.3.1	Bach-like generation	53
4.3.2	The Beatles-like generation	55
5	Conclusions and further work	56
5.1	Conclusions	56
5.1.1	Accomplished or partially accomplished objectives	56
5.1.2	Pending objectives	56
5.2	Further work	57
	References	58

List of Figures

- 2.1 Relation between notes and frequencies and their position in a keyboard. The first row of name-frequency pairs represents black keys, while the second row represents white keys. Source: <https://newt.phys.unsw.edu.au/jw/graphics/notes.GIF> 20
- 2.2 Grades: diatonic tones in a C Major scale. Source: <https://www.earmaster.com/images/misc/diatonicscale.gif> 23
- 2.3 Note and rest symbols and duration in beats. Source: <https://dannywalker9.files.wordpress.com/2013/06/06notevalues.png?w=650> 24
- 2.4 Music N releases structured in a tree of predecessors. Source: http://www.musicainformatica.org/wp-content/uploads/2011/09/music_n1.png 25
- 2.5 Note codes in MIDI. Source: <https://i.pinimg.com/564x/73/1f/77/731f7741cd7a26e3be6a131044592446.jpg> 27
- 2.6 Note resolution in MIDI. Source: http://mido.readthedocs.io/en/latest/_images/midi_time.svg 28
- 2.7 Gradient descent of a mean square error function (MSE) through 4 iterations. Source: https://cdn-images-1.medium.com/max/800/1*UUHvSixG7rX2EfNFTtqBDA.gif 30
- 2.8 A simple perceptron. Source: <https://cdnpythonmachinelearning.azureedge.net/wp-content/uploads/2017/09/Single-Perceptron.png?x31195> 31
- 2.9 Sigmoid and tanh activations (soft) versus sign activation (hard). Source: https://www.researchgate.net/figure/shows-the-activation-functions-sigmoid-fig3_285458781 33
- 2.10 Multi-layer perceptron (MLP). Source: https://www.researchgate.net/profile/Mohamed_Zahran6/publication/303875065/figure/fig4/AS:371118507610123@1465492955561/A-hypothetical-example-of-Multilayer-Perceptron.ppm 34
- 2.11 Recurrent Neural Network. The right image represents an unrolled representation. Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png> 35

2.12	Standard LSTM Cell. The black square indicates a delay of a single time step. Source: http://www.deeplearningbook.org/contents/rnn.html	36
4.1	Train and test loss general curve profile	49
4.2	Generated melody matrix with two interleaved notes.	50
4.3	Train and test loss with Dropout: 0.25.	51
4.4	Generated melody matrix with dropout.	52
4.5	Loss when optimizer is changed to Stochastic gradient descent (SGD) and loss function to Mean Squared Error (MSE).	53
4.6	Count of subjects with and without musical background.	54
4.7	Responses to 'How pleasant the music sounds?' where (A) is a piece from a human composer and (B) from an AI for Bach's dataset. . . .	54
4.8	Responses to 'How pleasant the music sounds?' where (A) is a piece from a human composer and (B) from an AI for The Beatles' dataset. .	55

List of Tables

- 1.1 Human resources cost 16
- 1.2 Equipment. Direct costs. 17
- 1.3 Complete project budget 17
- 2.1 Conversion between Solfège and Letter notation of the seven main tones in occidental music 21
- 2.2 Common scale patterns centered in C (for simplicity). 22
- 2.3 Main MIDI Messages.Source: <https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message> 27
- 2.4 MIDI Frameworks in common programming languages. 28
- 4.1 Experimentation parameters and results 50
- 4.2 Experimentation parameters and results with dropout 52
- 4.3 Results for the comparative between Bach’s dataset musical pieces. . 55
- 4.4 Results for the comparative between The Beatles’ dataset musical pieces. 55

1. Introduction

The following document describes the design, development and experimentation made through an autonomous music composition module.

This project aims at exploring a way to ease the music composition process, by generating a new melody from a dataset of music files. This approach seems natural, as a professional would always compose from a set of musical references.

Both human and machine-based techniques have been explored, but there are a lots of questions yet to be answered when it comes to compositions based on Deep Learning (DL) techniques.

The project consists on five main tasks:

- Pre-processing of MIDI (Musical Instrument Digital Interface) files to extract desired events and information.
- Dataset construction using the previous processed MIDI files.
- Training of an Artificial Neural Network (ANN). For the purpose of this project, a Long Short-Term Memory (LSTM) architecture has been selected.
- Generation of the new composition and storing into a new the MIDI file.
- Quality testing.

The first chapter is divided into two sections. Section 1.1 covers the main reasons that drove the project. Section 1.2 presents the goals that had been set in order to consider the project as concluded. Section 1.3 contains a study about the Regulatory Frame of music composition based on Artificial Neural Networks where a legal analysis and applicable technical standards are discussed. Finally, section 1.4 contains the budget of the project and Socio-Economic applications where the implementation of the project could be helpful for the society.

1.1. Motivation

The motivation underlying this Bachelor Thesis relies in the idea of making the music creation process easier and more accessible to people.

Learning to compose music might take many years from a formal education procedure, as it requires several knowledge about music theory and analysis, music performance, ear training, orchestration and arranging, and music history. However, music genres as pop rely on some basic rules that can be quickly identified and could help to simplify the problem by regulating the composition framework.

The core idea is that the composition task could be easier if a simple, but pleasant to ear melody, would be generated by an autonomous system. The composer would not need to start the whole process from scratch and could work on that base melody.

In addition, anyone could enjoy their own compositions even without a solid musical background and have a quick start on composing and improvising melodies. If the perfect model of the composition task was defined, it would be even possible to generate the base of a new hit by selecting the appropriate sound tracks. New possibilities for music are emerging now and maybe music creation could be an innovative and a highly technologically demanding field in the future.

Since 2012, research has proved that complex and abstract problems such as image and speech recognition are best solved by Deep Learning (DL) techniques. Generation problems that used to be solved by Markov models can also be solved now by implementing an Artificial Neural Network (ANN) architecture.

Exploring ways to teach a neural network to inference rules beneath music composition is a challenge that could led to the base melody that would simplify the whole composition process.

The author of this project, that counts with an elementary education in music and Artificial Intelligence (AI), is widely interested in the union of art and technology. A challenge as composing music (generating music, from an AI perspective), is then appropriate.

1.2. Objectives

The project aims at composing a pleasant to hear melody, that could either be a part of a song itself or could serve as a base to compose further complex melodies.

To achieve the main goal, a data model must be designed as a collateral task. The design of a model that was understood for either technical or non technical people would be a great contribution to the music generation field.

Music composition through Deep Learning techniques is a young research field (it has emerged on this decade), with lots of details to be explored and defined. Another purpose of this project is to shed some light on the music generation field, e.g. data extraction, model simplifications, metrics, training and generating issues, etc.

It would also be interesting to determine if humans are able to differentiate AI compositions from human compositions through an adaptation of the Turing Test.

Great amounts of data and computational power are required to solve problems through Deep Learning. The experiments were made with limited resources, so a great performance is not a priority, though it is a desirable goal.

1.3. Regulatory Framework

1.3.1. Legal analysis in Artificial Intelligence

Defining legal regulations of autonomous systems is a pending task. No laws about Artificial Intelligence have yet been defined, so the fair and constructive usage of available techniques only depends on the ethical values of the developer/belonging corporation. The legal gap affects both general Machine Learning and Robotic fields, though the study will focus on AI specifically.

Even without a legal framework available, the concerning on ethical developments has driven to some research publications. To bring an example, Virginia Dignum [1] claims that the path to achieve a responsible usage of AI has two sides. On the one hand, laws must define who has the responsibility in case of failure. On the other hand a data regulation of both used and created data is mandatory. It

could be said that authorities are starting to define the second side of the path to a responsible AI, as the new General Data Protection Regulation (GDPR) has been approved this year at the Spanish context. This new GDPR defines customer rights about data treatment and privacy, among other aspects.

Any creative task carries the right of intellectual property with itself. Particularly in music, the Spanish law automatically provides intellectual property of the final composition to its authors, since the moment of its creation [2]. The intellectual property law grants the acknowledgment, economical retribution and other benefits to its authors.

There is, however, a regulatory gap in the definition of author, resulting of the contrast of technology versus legal evolution.

The Spanish Royal Academy considers an author as the 'person who has produced a scientific, literary or artistic work' [3]. However, in music generation several authors might be identified:

- Even though it would be more precisely to consider the author as the autonomous system (as it is autonomous), there is no regulation at the moment that grants any kind of rights and obligations to an autonomous system, as the definition of author refers to a human being.
- The development team has the intellectual property of the script (or even platform) that performs the generation. Thus, reserved rights on submitted data and generated melodies depend on the code license.
- It is not clear if the final user would have the intellectual property of the generated piece. However, if the piece is modified in a way that it could not be related to the original piece, the user would have the intellectual property of the derived piece for sure.

1.3.2. MIDI as a technical standard

MIDI is the technical standard used in the project for music representation. It was standardized in 1983 'by a consortium of musical equipment manufacturers (including Korg, Oberheim, Roland, Sequential Circuits, and Yamaha).' [4]. The MIDI standard includes three different standardizations: first, there is a protocol standard for digital and electronic instruments communication. Second, the General MIDI specification defines the relation of instrument types with the different patch

or program numbers of a synthesizer. Finally, there is a specification for Standard MIDI Files (with .mid extension). Further information about the MIDI format can be found in section 2.3.2.

The US Copyright Office ("USCO") accepts copyright registration filings for original music in Standard MIDI File (SMF) format [5]. This means that, even when the melody is in the public domain, the MIDI file can be subjected to copyright protection. This can lead to the payment of royalties when commercial purposes are intended. However, in the case of music generation, the original MIDI files are just processed to extract relevant information, so the MIDI file can be perfectly used if it is public.

1.4. Socio-economic Environment

1.4.1. Budget

A virtual scenario is assumed in order to elaborate the project budget. Thus, the project duration, worked hours and roles are derived from the Bachelor thesis plan.

The project duration has been set to sixteen weeks, as the Bachelor thesis is designed to be completed in four months. Twelve ECTS are assigned to the Bachelor thesis and each one of them represent 25 hours for the student to work. It would take 300 hours of work for a developer to implement and evaluate the system: $12ECTS \cdot 25hours = 300$ hours of work. Assuming a temporal guard interval of 5%, total hours amount ascend to 450 hours.

Assuming weekly meetings of 2 hours with the Supervisor, this role would dedicate 32 hours to the project; 33.6 hours with guard intervals. (see Table 1.1 for a global perspective of the human resources).

Table 1.1: Human resources cost

Category	Cost (€ per hour)	Total hours	Total Cost (€)
Developer	15	450	6750
Supervisor	20	33.6	672
Sum			7422

As all software used has been free and open-source, the only direct costs considered has been related to the equipment.

About the equipment, only one laptop have been used with the following specifications: i7-6700, NVIDIA GeForce GTX 960M, 2GB, 16 GB RAM. The depreciation of the equipment has been calculated by following the next expression: $(A/B) \cdot C$ where:

- A: Period of use (months).
- B: Pay back period (months).
- C: Equipment unit cost.

Chargeable cost of the laptop can be checked in the direct costs table: Table 1.2.

Table 1.2: Equipment. Direct costs.

Equipment	Unit cost (€)	Months	Payback (months)	Chargeable cost (€)
Laptop	1099	6	48	137.38
Software	0	-	-	0
Sum				137.38

Indirect costs have been quantified as a 20% of direct costs, as there is no way to take into account Internet, water, light and gas bills for a theoretical project.

Table 1.3 contains the complete project budget, which has a total cost of 7586.86 €.

Table 1.3: Complete project budget

Total costs	
Human resources	7422 €
Direct costs (equipment)	137.38 €
Indirect costs	27.48 €
Sum	7586.86 €

1.4.2. Socio-economic impact

The development of this project can be helpful for the society in different situations:

A commercial version of the project could be used as a software in any recording studio, in order to compose the next hit. They could take advantage of the artist discography and new hits for the system to extract their characteristics and compose a novel one, with both last trends and the artist essence. In addition, the creation process could speed up, having a decreasing effect for total costs of the company.

In education, teachers could use the system for a wide variety of tasks:

- To illustrate concepts music theory related concepts, such as scales and harmonization. The teacher could generate as many examples as needed and the student could experiment by his/her own. The project could contribute to make more dynamic and collaborative music classes, where the student could see music composition almost as a game.
- To make it easier to memorize concepts: a simple song could be generated and the teacher could compose some lyrics about the topic. This can be applied to any subject, from elemental calculus to history.

Last but not least, people with short musical background could enjoy to compose melodies without the limitations of the high sloped learning curve of music theory. This could led to a democratization where anyone could express emotions by music.

2. State of the Art

Music is the art of combining sounds in a temporal sequence, applying different rules that provide mathematical relations on rhythm, melody and harmony. These rules generate patterns; combined sounds that produce expressive effects as tension, expectancy, and emotion on a listener.

As the project aims at implementing an autonomous system to inference those rules, extracting representative information is mandatory in order to train it with a good dataset.

Therefore, the first subsection will dive into the state of the art of Music Information Retrieval (MIR), to define what kind of information must be extracted and how to achieve that goal. Music concepts related to the information to be extracted are explained on subsection 2.2 to ease the understanding of the system design and its implementation. Section 2.3 will describe the existing computational methods to extract the required data. Finally, most common machine learning techniques for generative models are described in section 2.4, focusing on neural networks and the best architectures for generating music.

2.1. MIR Overview

By merging Downie's [6] and Meurer's [7] publications, Music Information Retrieval can be defined as the synthesis of multiple aspects of music representation, whether it is raw audio (e.g. MP3, OGG, AAC, PCM, AIFF, etc.), symbolical, (e.g. MIDI, MusicXML, score, etc.), or both.

As an expansion of text-based Information Retrieval, it aims to provide complete access to the information that any form of digital music representation contains and to reduce the gap between high-level musical information and low-level audio data.

The field of MIR unifies related disciplines such as acoustics, psychoacoustics, audio signal processing, computer science, musicology, library science, pattern recognition and machine learning. MIR applications extend to 'searching, extensive sorting, music recommendation, metadata generation, transcription, and even aiding/generating real-time performances' [7].

As Downie defines in an older publication, music information can be classified in seven facets: 'pitch, temporal, harmonic, timbral, editorial, textual, and bibliographic facets' [8]. The following paragraphs will focus on the analysis of music theory on pitch, harmonic and temporal facets, as no bibliographic or editorial content will be extracted from meta-data and there is no relevant information for composing in timbral aspects.

2.2. A brief on music theory

2.2.1. Pitch: Notes

A note is the audible signal generated by the air vibration at a constant and audible fundamental frequency (see the representation of a keyboard in figure 2.1). A note is also known as the minimum element of a musical sound.

Occidental music defines a system of twelve periodic notes to describe the audible frequency range. These intervals are known as octaves. Its definition is basic to understand notes nomenclature, as the thirteenth note will share the same name as the first, as can be seen in the keyboard in figure 2.1. A technical explanation could be found in the frequency domain; there is a relationship between the frequency of a note and that same note an octave above or under: the upper octave of a note will double its frequency, while same note in the octave below will be half its frequency.

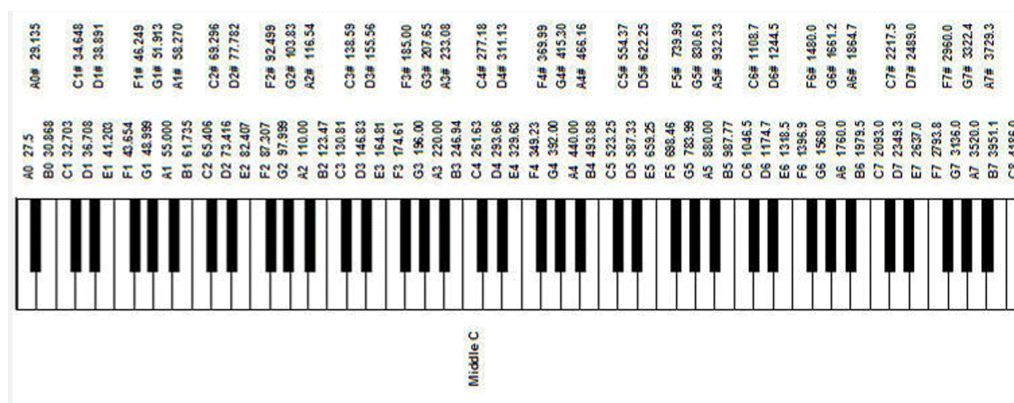


Figure 2.1: Relation between notes and frequencies and their position in a keyboard. The first row of name-frequency pairs represents black keys, while the second row represents white keys.

Source: <https://newt.phys.unsw.edu.au/jw/graphics/notes.GIF>

This system was primary developed in the middle Ages to make it easier to compose, learn and sing melodies. At first, notes were named by the first letters of the alphabet, which is an equivalent to a later notation: Solfège. Nowadays both notations coexist, though letter notation is typically used in English-speaking countries. The correspondence between both notations can be found in Table 2.1

Table 2.1: Conversion between Solfège and Letter notation of the seven main tones in occidental music

Letter notation	Solfège
C	Do
D	Re
E	Mi
F	Fa
G	Sol
A	La
B	Si

As showed in figure 2.1, both notations give names to the notes represented by the seven white keys on the piano, leaving related names for the five black keys even though they are equally important from a frequency perspective. Thus, the higher black key of A will be A sharp (\sharp) and the lower black key will be A flat (b).

This representation is made for convenience, as some notes are more likely to be paired with a selected group of notes than with others in a composition as we will see in the next section.

2.2.2. Harmony: Scales and Intervals

A *scale* is an organized sequence of notes; a selection of notes that share harmonic properties as a common pattern represented as the intervals between two subsequent notes measured in semitones within an octave. *Intervals* define the distance between notes e.g. as explained before, an octave is an interval of 12 notes.

The most common scale is C Major; a scale centered in C that follows the pattern: 2-2-1-2-2-2-1 (a Major pattern). It is so important that the keys of the piano were designed following that pattern; looking at a C note in figure 2.1, the pattern represents all the white keys: C, D, E, F, G, A, B, so that they would be easy to find.

More examples can be found in the table 2.2, which contains examples of typical scales and their patterns.

Table 2.2: Common scale patterns centered in C (for simplicity).

Scale	Pattern
Minor C	2-1-2-2-1-2-2
Dorian	2-1-2-2-2-1-2
Phrygian	1-2-2-1-2-2
Minor harmonic	2-1-2-2-1-3-1
Pentatonic	2-2-3-2-3
Whole tone scale	2-2-2-2-2-2
Chromatic	1-1-1-1-1-1-1-1-1-1-1
Blues	3-2-1-1-3-2

Besides the aforementioned *octave*, another basic interval is a major second: a distance of a tone or two semi-tones, or notes, as were named earlier. Referring to tones and semitones makes it easier to relate notes to keyboard representation; a distance between a black key and a white key is a semi-tone and the distance between white keys is a tone. There is an exception, though: the pair of white keys with no black-key in the middle have a distance of a semi-tone.

To describe other intervals between notes, it is important to note that there are two strands that define the same intervals with different names: the diatonic and the chromatic perspective. In order to keep a simple explanation, it could be said that a diatonic interval expresses distance in white keys. A chromatic perspective also includes black keys in the counting. This document, in particular, will follow a diatonic notation when it is needed.

Tones represented in figure 2.2, are numbered following their positions in the C Major scale: C = 1, D = 2, E = 3, etc. Referred to C, an interval of second would refer to a transition from C to D, whether it is D# or Db, an interval of third to a transition from C to E, etc. In this case, this notation is equivalent to degree notation, where G would be the 5th degree of C Major scale. When referring to degrees, the *tonic* is the first note of the scale (C in the previous example).

However, the sound transitioning from C to D is not the same than the sound from C to D#; these intervals have different qualities. The quality of an interval can be Perfect, Diminished, Augmented, Major, or Minor.

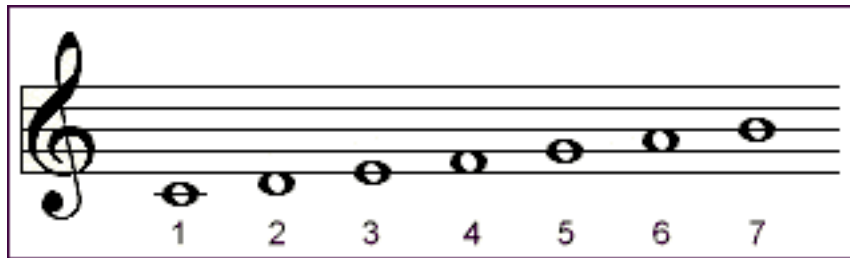


Figure 2.2: Grades: diatonic tones in a C Major scale.

Source: <https://www.earmaster.com/images/misc/diatonicscale.gif>

Perfect intervals are fourth, fifth, octave and unison (same tone). All of them can be augmented, if they are one semi-tone larger, or diminished if they are one semi-tone smaller. The rest of the intervals (second, third, sixth and seventh) can be major or minor, where minor means that the interval is a semi-tone smaller than the major version of the same interval. There are also augmented and diminished seconds, thirds, sixths and sevenths but they are not so commonly used.

2.2.3. Temporal: Notes, rests and tempo

In a staff representation, the previous pitch-related concepts are shown in a vertical position, while time is represented in horizontal. Time is a fundamental part of a piece of music, as rhythm is achieved with periodical sequences of notes, delimited by bars in the staff.

The duration of a note, or a rest, is determined by standardized symbols with a formalized relation (see figure 2.3). These relations, however, do not quantify duration in time units such as seconds or milliseconds. Historically, time references were taken from tempo marks at the beginning of the music piece.

There is a correspondence with an indirect meaning in time units: the beat. A beat is a measure that depends on the time signature of the piece. For example: a typical 4/4 time signature refers to a beat as a quarter note and indicates that a complete bar is formed of four beats. The standardized term for measuring the tempo of a piece of music is beats per minute (bpm), e.g. modern music used to be composed at 120 bpm, though pop music is lately dropping to 90.5 bpm.








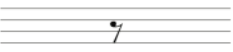

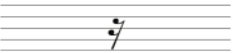
	Whole Note 4 Beats	
	Half Note 2 Beats	
	Quarter Note 1 Beat	
	Eighth Note $\frac{1}{2}$ a Beat	
	Sixteenth Note $\frac{1}{4}$ of a Beat	

Figure 2.3: Note and rest symbols and duration in beats. Source:
<https://dannywalker9.files.wordpress.com/2013/06/06notevalues.png?w=650>

2.3. Computational music description

While symbolic and spoken representations of music in live performances have a high level of abstraction (even an emotional component), computer representation of music must be mathematically precise and concrete.

According to Dannenberg [9], different levels of abstraction must be defined, as each level of abstraction synthesizes unavailable information in other levels. Thus, three levels of abstraction must be distinguished:

High level: printed notation, as it is symbolical and abstract.

Intermediate level: emotional information in a live performance. E.g.: to raise the brightness of a note means to appropriate tune a note that could sound flat.

Low level: digital audio signals and other bit-based representations, as they must remain non-symbolic and concrete.

Focusing in low level representations, several software, codecs, and languages have been developed to represent and work with music. However, working with raw audio was undesirable as audio signal processing was required to extract information about the three facets: pitch, harmonical and temporal. The document will not analyze available audio formats or digital audio workstations as they were not considered for the system design and implementation. The analysis will focus on two music languages instead: CSound and MIDI

2.3.1. Music N (1957 - 1986) and CSound

As discussed in [10], Music N is a family of computer music languages made by different organizations (e.g. Bell Labs, Princeton University, Stanford University, M.I.T., etc.) that share some characteristic features.

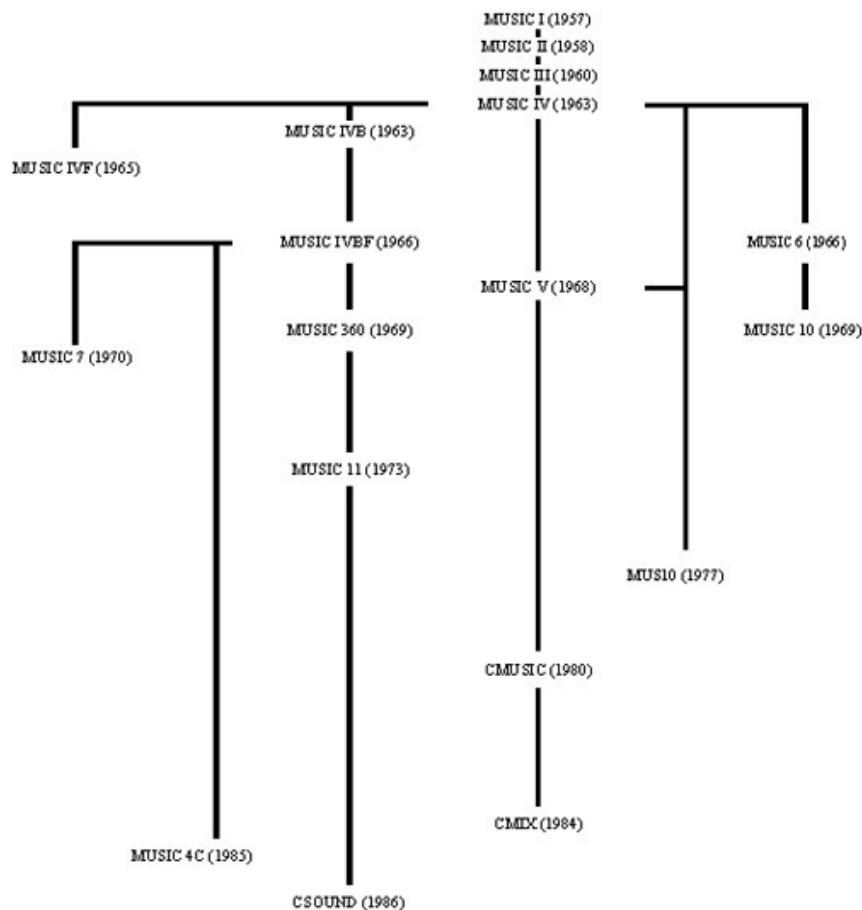


Figure 2.4: Music N releases structured in a tree of predecessors. Source: http://www.musicainformatica.org/wp-content/uploads/2011/09/music_n1.png

The design of this set of languages has influenced later developments, whether they do or do not belong to the family of Music N. Different versions were launched over a period of forty years, where early versions were developed in Assembler and Fortran, finally evolving to C.

Music I and its predecessors define musical parameters with letters and numbers, enabling to compile a text file without graphical interfaces, even though through the years graphical approaches to composition have also been developed.

Since Music III, the usage of units generator has been extended to later designs and has had a big impact on computer music in general. Units generator

are useful macros to generate or control sound, such as delay, oscillators, filters, amplitude envelopes, etc. The composer could then concatenate several units to create its own instruments.

One of the main disadvantages of the Music N family is the deferred time processing attached to hardware limitations of that time. This issue made impossible to generate real-time compositions. Only the last version, CSound (1986), could operate in real time.

CSound development has continued from the mid-eighties until today, expanding its fields of application by creating a whole ecosystem to create music. Nowadays, it is used by composers and musicians for any kind of music that can be made with the help of the computer. As an open-source platform, a broad community of volunteers take part on its development with bug reports, feature requests and discussions and provide examples, documentation and articles [11].

Even though it can be used as a framework from a wide variety of programming languages (e.g. such as Python, Lua, C/C++, Java, etc.), the high complexity and learning-curve of installing a full environment combining CSound and Machine Learning libraries made preferable to simply use MIDI to translate from bits to music.

2.3.2. MIDI

MIDI refers to Musical Instrument Digital Interface. It is an standard protocol for communications between many different kinds of devices such as digital instruments, cell-phones, computers, and audio devices, even though it was originally designed for communicating several synthesizers. As an standard, its applications extend to music production, personal computers and television, as it is great for developers, composers, producers, educators, and for domestic purposes.

The protocol operates by MIDI Messages. They describe which notes are to be played and for how long, as well as the tempo, which instruments are to be played, and at what relative volumes. [12] Most common MIDI Messages, its description and code are shown in table 2.3.

Each MIDI note has some defined properties:

Key: Note code in MIDI standard.

Table 2.3: Main MIDI Messages. Source: <https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message>

Status	Data Bytes	MIDI Message	Description
D7—D0	D7—D0		
1000nnnn	0kkkkkkk 0vvvvvvvv	Note Off event	Sent when a note is released (ended). kkkkkkk: key number vvvvvvvv: velocity.
1001nnnn	0kkkkkkk 0vvvvvvvv	Note On event	Sent when a note is depressed (started). kkkkkkk: key number vvvvvvvv: velocity

Velocity: Quantifies the intensity of the note; the strength that has produced it.

Time: Relative passed time since the last event.

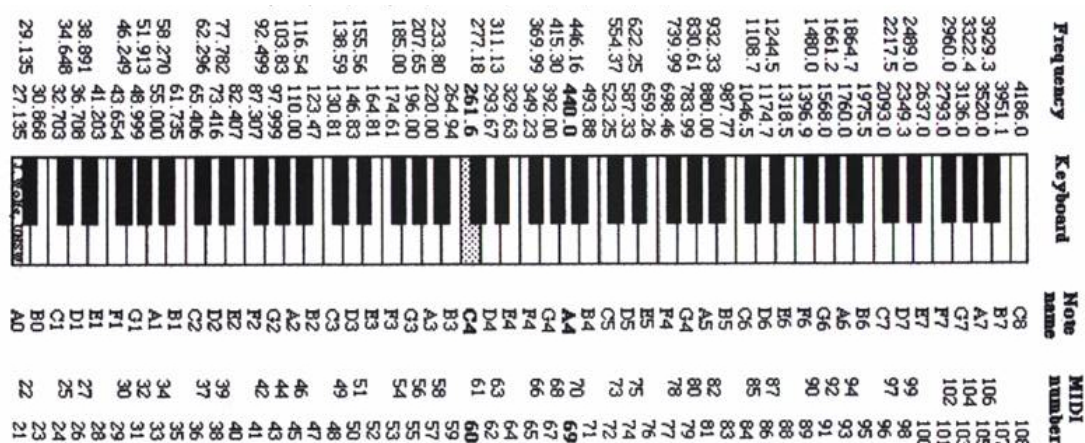


Figure 2.5: Note codes in MIDI. Source:

<https://i.pinimg.com/564x/73/1f/77/731f7741cd7a26e3be6a131044592446.jpg>

Inside a MIDI track, tempo is characterized by MIDI ticks; a magnitude that provides resolution to beats per minute (defined in section 2.2.3). Thus, a resolution of an sixteenth note will be achieved with 4 ticks per beat. Figure 2.6 shows a graphical example of a resolution of 3 ticks per beat.

MIDI is not an audio recording per se, just a sequence of instructions, which the device that performs the music piece will interpret, in the form of MIDI Messages. Thus, the way a MIDI file sounds depends on the finally used device or on a previous synthesizing stage.



Figure 2.6: Note resolution in MIDI. Source:
http://mido.readthedocs.io/en/latest/_images/midi_time.svg

The protocol counts with some of advantages derived from its not-audio nature: storing generated sounds in a MIDI file will require of much less storage than a real audio file (typically 10 Kb per minute). This has a directly effect on the processing speed of the file.

There is also a wide variety of frameworks designed to work with MIDI in common programming languages. Python and Matlab must be highlighted from table 2.4, as they are better for prototyping tasks.

Table 2.4: MIDI Frameworks in common programming languages.

Language	Frameworks available
Python	Mido, python-midi,
Matlab	Audio System Toolbox TM
Java	MIDI package

2.4. The role of machine learning for music generation

Machine Learning (ML) aims at designing general purpose and efficient algorithms of practical value, to teach computers to solve tasks such as accurate classifying or predictions. In ML ‘a large set of N digits $\{x_1, x_2, x_3, \dots, x_N\}$ called a training set is used to tune the parameters of an adaptive model’ [13]. Thus learning is always based on the convergence (training) of a general function that maps a set of observations (data samples).

Generalization is so important that there is always a test set of data, different from the training set, to evaluate the effectiveness of the machine learning

algorithm by counting how many of the outputs of the samples are correctly computed by the method [14].

ML is included as a sub-area of Artificial Intelligence, though it also intersects broadly with other fields, especially statistics, but also mathematics, theoretical computer science, etc. [15].

A wide variety of techniques have been designed to achieve that goal, such as Decision Trees, Naive Bayes, Genetic Algorithms, Markov Chains, Support Vector Machines, Neural Networks, ... However, the document will focus on techniques specially designed for generating sequences because of the project scope. An exhaustive analysis of the state of the art is not intended, as it would be unmanageable.

ML-based successful applications are for example spam detection, tumor detection, text generation, language translation, image caption generation, object classification, game playing or handwriting generation [16].

Two different broad classes of procedures exist:

Supervised learning: the learning stage counts on labeled data. The model is tuned for each iteration to fit the targets, i.e, the labels.

Unsupervised learning: the learning stage does not count on labeled data. These tasks often consist on data clustering.

Music generation has been broadly studied from a Machine Learning perspective over the past decade. Past related projects often use Genetic Algorithms [17] and Markov Chains [18] to generate music. Complex hybrid models of Restricted Boltzmann machines and Recurrent Neural Networks have been developed [19] in order to model unconstrained polyphonic music. However, these methods do not achieve a successful modeling of longer-term musical structure.

Huge advances in accuracy and speed were obtained by Deep Learning around 2012. Nowadays it is possible to model long-term structures in generation tasks, as an specific deep neural network architecture tends to be suggested: Long Short Term Memory architectures (LSTM).

Some machine learning terminology needs to be defined in order to understand and enhance the training performance of our system:

Cost function models the cost of the error of the classification or prediction task. This term is equivalent to loss or error function.

Over-fitting is an undesirable effect that happens when the the function fits really well the training set, but it fits badly (or not as well) the test set. In other words, the cost or loss will decrease with a different behavior in train and test.

Gradient descent is an iterative optimization algorithm that finds the minima of a curve. In machine learning it is used to set parameters (e.g. weights) minimizing the cost function in each iteration. The learning rate defines how big the step of the gradient descent will be in a determined iteration of the algorithm. In some frameworks as Keras, algorithms as Gradient Descent are referred as *optimizers*, as they try to find the absolute minima of the cost function.

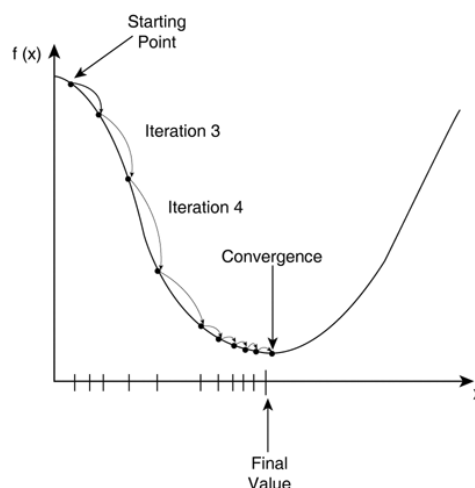


Figure 2.7: Gradient descent of a mean square error function (MSE) through 4 iterations. Source:

https://cdn-images-1.medium.com/max/800/1*UUHvSixG7rX2EfNFTtqBDA.gif

Batch size: defines the number of input instances that will be fed and propagated through the network. E.g. if the batch size is set to 128, that number of input instances will be fed to the network each time. Different training techniques are defined depending on the existence of batches:

- Stochastic Gradient Descent (SGD): calculates the error and updates the model for each input instance of the dataset. Even though it is computationally expensive, it often is a good solution to avoid the loss to get stuck in a local minima.

- Batch Gradient Descent: calculates the error for each input instance of the dataset, but the model updates are performed at the end of each epoch. It is more computationally efficient than SGD but sometimes leads to the convergence of the model in suboptimal parameters.
- Mini-batch Gradient Descent: the dataset is fragmented in small batches. The error is quantified and the model updated for each mini-batch. It is an intermediate solution that finds the balance between robustness and efficiency, which makes it a common technique in Deep Learning.

The following paragraphs will explain all required concepts to broadly understand LSTM, by starting with Neural Networks fundamentals, diving into Deep Learning to finally explain LSTM functionality.

2.4.1. Artificial Neural Networks

A neural network is a computational model inspired by the biological process of information 'where hundreds of billions of interconnected neurons process information in parallel' [20]. A high number of processing units (neurons) produce linear outputs in each iteration of the algorithm, in order to fit a series of targets (labels).

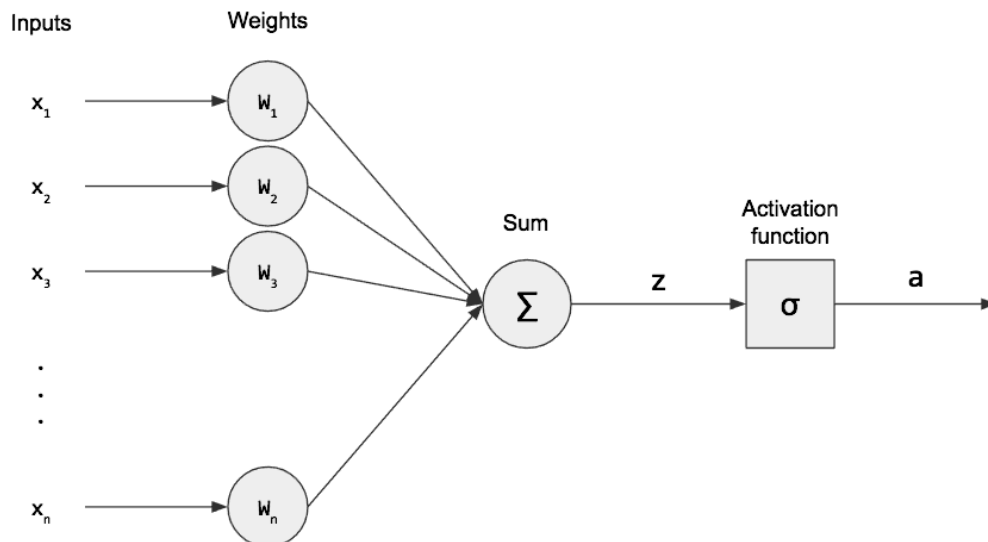


Figure 2.8: A simple perceptron. Source: <https://cdnpythonmachinelearning.azureedge.net/wp-content/uploads/2017/09/Single-Perceptron.png?x31195>

In figure 2.8 the schematics of a perceptron is shown. This unit is a simplified mathematical model of how the neurons in our brains operate. The perceptron

estimates an output given a set of inputs, a set of weights, a bias and a threshold that is usually included in the bias. Weights and bias are typically initialized to small random numbers.

Perceptron Learning Algorithm In its original formulation, all inputs are multiplied by its weights and summed. The activation function, the sign function, sets the threshold for final output to be binary. In each iteration:

$$a = 1 \text{ if } \sum w_n x_n + b > 0$$

$$a = -1 \text{ if } \sum w_n x_n + b < 0$$

In order to learn, the output a is compared to the correspondent desired value d , if they are not equal, weights must be updated by using the following algorithm:

- If the target and the output are equal, the weights remain with the same value.
 $w(t+1) = w(t)$
- If $d = 1$ but $a = -1$: $w(t+1) = w(t) + \alpha x(t)$
- If $d = -1$ but $a = 1$: $w(t+1) = w(t) - \alpha x(t)$

α is a number between 0 and 1 equivalent to the learning rate. These steps are repeated until the classification error is satisfactory or a given number of iterations are completed. [21]

A perceptron actually performs a binary classification by the approximation to a linear function, as inputs and weights represent a linear combination.

A basic system to perform predictions is the Adaline (ADaptative LINear Element). The architecture is equivalent except for the activation function, which is the identity in this case. This system estimates linear functions as linear regressions. The learning expression for the Adaline comes from the gradient descent of the difference between the output and the desired value. In the following expression, τ is a variation rate:

$$w(t+1) = w(t) + \tau e(t)x(t) \text{ where the error expression is: } e(t) = d(t) - w(t)x(t)$$

These systems are designed to classify or predict linear functions, but they fail with non-linearities even when lots of neurons are added to the learning process. Non-linearity issues were solved by adding more layers to the architecture, but there was no way to find an expression to update the weights of the internal (hidden) layers. It was also proved that there was no guarantee for the algorithm to converge, so the research remained idle for fifteen years.

The main problem was that the activation function of the perceptron was not differentiable, so there was no way to perform a gradient descent of the error to find a new set of weights. Thus the solution consisted on choosing a differentiable activation function; for example, a sigmoid or a tanh function.

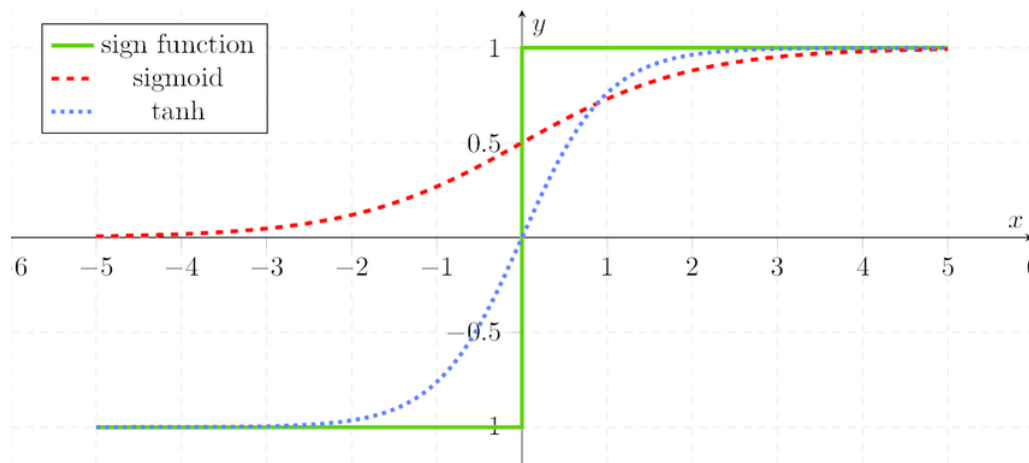


Figure 2.9: Sigmoid and tanh activations (soft) versus sign activation (hard).

Source: https://www.researchgate.net/figure/shows-the-activation-functions-sigmoid-tanh-and-the-sign-function_fig3_285458781

A multi-layer perceptron architecture (MLP, as shown in figure 2.10) with soft activations would then be able to model non-linear functions by finding the internal weights updates applying the chain rule of the gradient descent to previous layers (backpropagating the error from the output layers towards the input ones).

The MLP architecture has a fundamental advantage: it is able to establish any kind of input-output correspondence; having the capacity to solve any problem. There is no reference, though, to the number of units or hidden layers needed to solve it and huge amounts of data and high computational power are required for them to have an optimal behavior.

2.4.2. Deep Learning

At the late eighties, the term Deep Learning started to refer to feature extraction systems with many layers and non-linear activation functions. Deep Learning architectures allow to model high level and abstract models by adding layers, being adequate for tasks such as image and speech recognition.

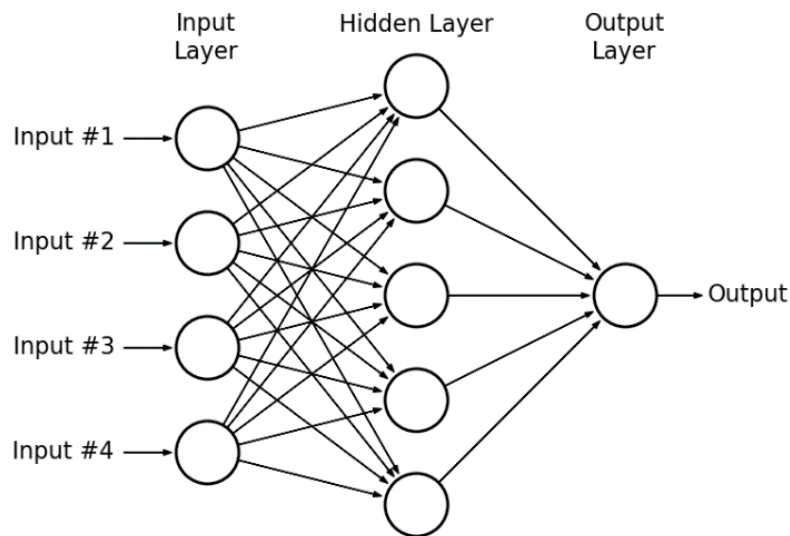


Figure 2.10: Multi-layer perceptron (MLP). Source:

https://www.researchgate.net/profile/Mohamed_Zahran6/publication/303875065/figure/fig4/AS:371118507610123@1465492955561/A-hypothetical-example-of-Multilayer-Perceptron-Network.ppm

With the raise of Big Data and the usage of GPUs, it is no surprise that the research on Deep Learning architectures encompasses big parts of the state of the art of a wide variety of tasks.

A few important terms are explained in the following paragraphs, just as it was done in parent section 2.4.

In any ANN architecture an epoch defines the entire dataset passage forward and backward through the layers of the network. The number of epochs then defines how many times the entire dataset is gonna be fed to the network.

Dropout is a common applied technique when overfitting is detected.

Deep Neural Networks result very expressive models, as they are able to learn complicated input-output relations. This means that complex relations might appear in the training set but they might not appear with novel testing data. The problem gets worse if a limited set of data is provided for training, as some of those complex relationships might be the result of sampling noise. This scenario is a typical case of overfitting in DL.

Dropout consists on switching off (dropping out) units; temporarily disconnecting the unit from the network, as well as all its incoming and outgoing connections. Dropout also provides of major improvements over other regularization

methods. [22]. Dropout follows a random distribution where a neuron is present with a determined probability. Dropped units might belong to either hidden or visible layers.

2.4.3. Recurrent Neural Networks

A particularly interesting architecture was design to perform tasks that require persistence (e.g. text generation, translation, speech recognition, image captioning, etc.): Recurrent Neural Networks (RNNs) made the processing of past events as input data in order to predict new ones possible.

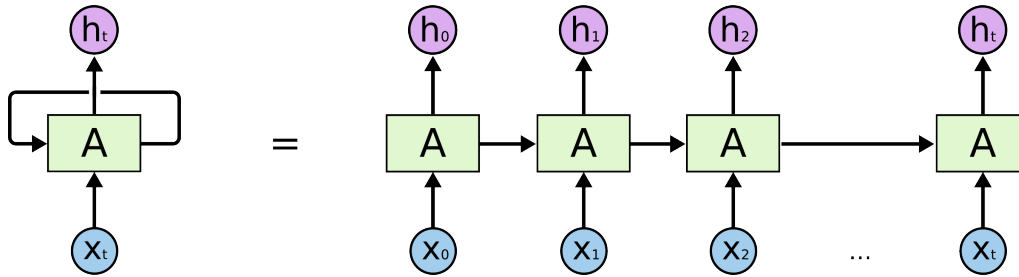


Figure 2.11: Recurrent Neural Network. The right image represents an unrolled representation. Source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png>

RNNs are basically neural network architectures with loops that allow information to persist by sharing parameters between different time steps. In figure 2.11, the perceptron A receives a series of inputs $X_t = \{X_0, X_1, X_2, \dots, X_N\}$ in each time step and makes a copy of each of them to take them into account at the future next time steps. Thus, the architecture is designed to process sequences and lists [23]. Most recurrent networks can also process sequences of variable length.

The unfolding process makes it possible to learn a single model that operates on all time steps and all sequence lengths, rather than needing to learn a separate model for all of the possible time steps [24].

The learning procedure for RNNs is an algorithm called *Back Propagation Through Time*. A detailed explanation can be found in Werbos publication [25].

Problems related to music require such a long persistent context that RNNs cannot successfully connect the required past information with present data. The technical explanation relies in the fact that linear gradients propagated over many stages tend to 'either vanish (most of the time) or explode (rarely, but with much

damage to the optimization)’ [24], as a composition of many nonlinear functions is performed. Thus, another kind of architecture, based on Recurrent Neural Networks is needed: Long Short Term Memory networks.

2.4.4. Long Short Term Memory networks (LSTMs)

Long Short Term Memory networks are a special kind of Recurrent Neural Networks that are capable of modeling long dependencies between data. While Recurrent Neural Networks units consist in simple tanh activations, LSTM architectures consist of LSTM cells. A LSTM cell has four internal paths (some even consider them as layers), as can be seen in figure 2.12: inputs, input gate, forget gate and output gate.

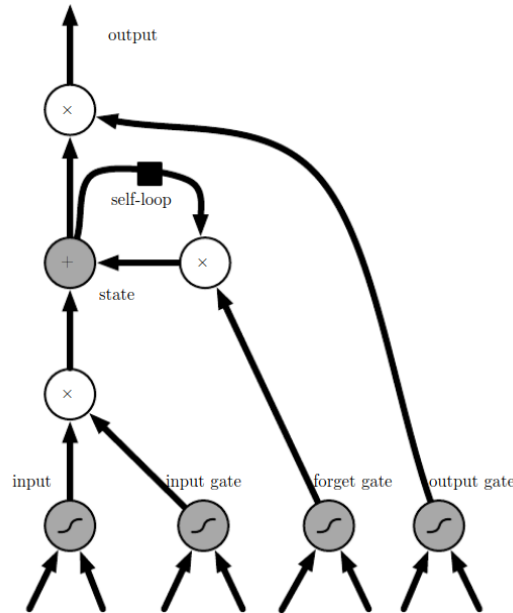


Figure 2.12: Standard LSTM Cell. The black square indicates a delay of a single time step. Source: <http://www.deeplearningbook.org/contents/rnn.html>

The idea is to provide gates to the system that have derivatives that neither vanish nor explode, allowing the gradient to flow for larger periods of time and protect it from undesired perturbations. The LSTM decides in each time step when to store past information in its cell state and when to forget it by the usage of different gates with sigmoid activations.

As shown in figure 2.12 and briefly discussed in [26], the characteristic loop of the RNN is controlled by the forget gate, that 'learns to reset a memory cell when its content is obsolete'. The external multiplicative input gate 'learns to protect the

flow from perturbation by irrelevant inputs'. The output of the LSTM cell can be deactivated by the sigmoid of the output gate which 'learns to protect other units from perturbation by currently irrelevant memory contents'.

As also defined in [26], the learning procedure of a LSTM consists on the 'Gradient Descent method that uses a slightly modified, truncated BPTT (Back Propagation Through Time) and a customized version of RTRL (Real Time Recurrent Learning). Output units use BPTT; output gates use the truncated version of BPTT; while weights to cells, input gates and forget gates use truncated RTRL'.

An important fact to keep in mind must be explained continuing with the Dropout analysis of section 2.4.2: unfortunately, a standard usage of Dropout does not work well with Recurrent Neural Networks. However, 'when correctly used, it greatly reduces overfitting in LSTMs' [27].

As Dropout 'can be interpreted as a way of regularizing a neural network by adding noise to its hidden units' [22], recurrent connections amplify that noise leading to a bad performance of the network. Thus Dropout must be applied to non-recurrent connections to improve the network performance.

2.5. Related works

The following sections will describe music generation projects and publications based on similar techniques and generation from symbolic music. The combination of data extraction from MIDI files and Long Short Term Memory networks is quite common in music generation tasks. The way music data is modeled, however, tends to change between projects. It might be noticed that some ideas of these projects have inspired the final system design.

2.5.1. Magenta project

Magenta [28] is a research project that explores ways to create art by using machine learning techniques. It was first started by Google Brain engineers though a wide variety of professionals have already contributed to the project. All of their projects are published in their GitHub page and developed over TensorFlow.

Their projects mainly cover painting and music art. Music, in particular, has been widely explored as they have already developed music generators, synthesizers

and collaborative duets between humans and AI. They also provide an interesting framework to convert a dataset to different audio formats with TensorFlow such as raw audio, MIDI, MusicXML, etc.

Magenta has both pre-trained and un-trained models to generate music, both of them based on LSTMs. This way, both technical and non-technical people can use the technology to compose music.

All of these models generate music based on language modeling. They face different aspects of music such as polyphony modeling, timing, dynamics, etc. The following list, extracted from (cite) contains all available Magenta models to generate music based on LSTMs.

- Drums RNN: generates drum track.
- Melody RNN: generates melodies.
- Improv RNN: generates melodies, conditioning them on an underlying chord progression.
- Polyphony RNN: generates polyphonic music. It is based on the BachBot architecture. BachBot [29] is a project that harmonizes music in the style of Bach.
- Performance RNN: generates polyphonic music. This can be achieved by extracting the following MIDI events: note on/off, timeshift, and velocity change events.
- Pianoroll RNN-NADE: generates polyphonic music. The LSTM is combined with a NADE, an architecture called an RNN-NADE [19].

2.5.2. Blues improvisation

Douglas Eck and Jurgen Schmidhuber implemented a system to generate improvisations of blues music over a sequence of fixed chords [26].

The data model was based on a one-hot encoding where 1.0 represented a note on and 0.0 represented a note off. Each note was an input instance with a related target

To model tempo, each input vector represented a slice of real time with a determined resolution per note. E.g. to represent an quarter note, four time steps were required for the network to process the whole note. This method was first introduced in [30] and was selected for its positive impact on timings.

This model had two approaches to ease data representation: first, it did not take into account when a note ended. Second, the dataset only contained ranges of 12 notes for chords and 13 notes for melodies.

As blues counts with a determined scale (see Table 2.2), a fixed sequence of chords was encoded so that a melody that fitted those chords was generated. The training set contained hand-crafted melody sequences, of quarter notes with no rests, that fitted the chords in every time step.

2.5.3. Konstantin Lackner's Composer

Konstantin Lackner [31] implemented a system to generate melodies given a set of chords of 16 The Beatles songs from the music book "Pop Classics For Piano: The Very Best Of The Beatles - Easy Arrangements for Piano".

The data model was similar to Eck and Schmidhuber approach, as the author claims 'to be motivated by their promising results'. In this case, the encoding was performed on two different matrices; one for melody and another one for chords.

The main difference between this encoding and the previous one was that note endings were also encoded. This allowed to differentiate between, e.g. two eighth notes and one quarter note on the same pitch.

This system used MIDI encoded on 96 ticks per beat, so a preprocessing stage sampled matrices to the equivalent of 8 ticks per beat. Reduced matrices dimensions were, however, duplicated to allow note off representations; each row of both matrices would represent half of a tick and the end of a note position would be encoded with the first half in 1.0 and the second half in 0.0.

An online test was made to know if subjects could identify the generated piece and if they preferred the A.I. composition better than the original one.

3. System design

A great amount of projects served as an inspiration to set our system. However, as music generation with machine learning in general, and artificial neural networks in particular, are relatively new research fields, a lack of documentation often occurs. This fact made it necessary to go back and try new approaches at some points of the development of the project.

Therefore, we will first describe our first generations with Magenta in section 3.1. However, since Magenta acts very much like a 'black box' when it comes to generate music, we decided to use a new design in order to fully understand how music generation worked with Long Short Term Memory networks. The new design is explained in section 3.2.

3.1. Initial approach

At first, Magenta's solution seemed quite convenient, as this project aims at providing a backend for experimenting and generating music in a short period of time.

The ecosystem for generations consisted on Python 2.7, TensorFlow GPU 1.5 and Magenta-GPU 0.3.1. In order to run Magenta, any of both versions of Python, Miniconda installer and extra packages as `rtmidi`, `build-essential` `libasound2-dev` and `libjack-dev` are required.

The dataset used for the first generations contained 227 MIDI files of classical music piano composers (e.g. Bach, Beethoven, Mozart, Chopin, Debussy, Tchaikovsky, etc.). All MIDI files can be found on the Classical Piano MIDI Page [32] and they are free to use and modify.

Magenta abstracts the whole process of data modeling, training and generating to four scripts of hyper-parameters; one for each step.

First, the set of MIDI files is transformed into a TFRecord file of NoteSequences; a file with extension `.tfrecord` that storages a kind of TensorFlow data-format called protocol buffers.

The next script extracts information contained in the TFRecord file to SequenceExample: each one of them contains a sequence of inputs and a sequence of labels that represent a melody. Two TFRecord files are generated: one for training and another one for testing. At this point, the train-test slicing of the dataset is performed by the adjustment of a parameter: eval-ratio.

In order to train and evaluate the model, two independent scripts must be executed with the same hyper-parameters. There are several properties to be settled at the train stage:

- Magenta model to choose.
- Batch size (default batches of 128 instances).
- Number of layers and number of LSTM cells for each layer.
- Number of training steps.

The training process is opaque as there is no way to change the internal settings of the LSTM layers (e.g. adding dropout, changing loss function, optimizers, etc.). In addition, there is no way to modify train and test execution traces.

The generations had good quality, but they sounded the same from one experiment to another. That issue, summed to a divergence between train and test loss led to think about a case of over-fitting.

After several generations of each model, it was clear that there were lots of constraints when using Magenta. The best way to understand music generation was to develop a full script that covered all steps.

3.2. The final chosen system

The final design simulates a very common technique to compose. Chords, the harmony line, are first extracted from the set of MIDI files and the system creates a melody that fits them, as used in [26] and [31]. This is inspired on a widely used composition technique where a determined sequence of chords is set and the melody is, then, constructed over the sequence with harmonic intervals.

The idea was taking advantage of this harmonic relation to generate a possible *matching* melody in a set of chords. In order to do it, the system should take the harmony line in each time step as input and the correspondent melody line of the next time step as targets.

In [33] a novel method to extract a melody from a polyphonic raw audio file is described. An accurate system to extract melodies is a hard related to audio signal processing and was out of the scope of the project. Thus, a pragmatic approach was considered:

As processed MIDI files could either present harmony and melody in the same or in different MIDI Tracks, two independent regions were established: as the harmony is used to be played with the left hand on the piano, and the melody with the right hand, the limit between harmony and melody would be middle C (C4). Thus notes to the left of C4 would be considered as harmony and notes to the right of C4, including C4, would be considered as melody.

At first only one octave of chords (from C3 to B3) and two octaves of melody (from C4 to B5) were considered. However, this did not seem to be enough; long silent periods arose in the processed data, as some notes were ignored. In later experiments, ranges expanded a whole octave. The final ranges were:

- Harmony (inputs): Two octaves, from C2 to B3 (24 notes)
- Melody (targets): Three octaves, from C4 to B6 (36 notes)

The following section will analyze possible implementations of each part of the project, with their advantages and disadvantages taken into account in order to get to the final decision.

3.2.1. Programming Language

As the project consist on a proof-of-concept to explore music generation with machine learning techniques, important requirements needed to be satisfied:

- Scripting and prototyping must be quick, as the project does not aim to develop a commercial product.
- It must be easy to process and extract MIDI data; it must count with a friendly MIDI framework.
- There should be available packages for mathematics and scientific computing
- Flexible Deep Learning frameworks must exist for that language, as they are a fundamental part of the process.

C [34] and C++ [35] were both possible options, as they are powerful and really quick on execution. C++ was plausible, as both TensorFlow [36] and OpenNN [?] frameworks are available for developing tasks based in neural networks in that language.

However, the author's C++ expertise is not too big and it seemed like most researchers use Python [37] or Matlab [38] for machine learning related tasks. The second fact was important, as frequent questions submitted to the Internet are often focused on those languages and they are a fundamental help for any personal project.

Matlab was a good option, as it is mathematically oriented and has packages for both MIDI and Deep Learning. However, some functions belong to non-free packages that are not included in the standard student license, and that was a risk that could not be taken.

Python is an open source and general purpose language widely used for prototyping in both industrial and academic fields. It has a simple syntax that it makes it easy to use, even for developers with just a slightly experience on the matter, as it was the case.

The use of Python in science projects is very extensive, as it offers highly attractive and widely known frameworks that tend to work together, such as Numpy [39], Matplotlib [40], Pandas [41] and SciKit-Learn [42]. For machine learning purposes we can find Tensorflow, Theano [43] and Keras [44], whose usage is widely extended for any of them. It also has some attractive MIDI frameworks published such as Mido and Python-Midi, both published in Pipy; the Python Package Index.

Among all programming languages, Python seemed to be the most flexible and convenient choice and was finally selected because of its simplicity, community and frameworks.

3.2.2. Frameworks

Lots of Python attractive frameworks have already been named in section 3.2.1. Comparison will be established between all of them but there are special cases.

Numpy is an extremely useful framework for scientific computing in Python and, if they exist, the alternatives do not have an extensive usage. Numpy allows the usage of a similar syntax to Mathematica based languages, making it suitable

for working with numerical data structures such as matrices and tensors. Therefore, it is one of the main frameworks selected were no alternatives have been taken into account.

The next comparative will led to the selected Deep Learning frameworks. Keras will be used to abstract some common tasks such as back propagation through time, to allow quick implementations of LSTM cells, etc. Thus, the real comparative must be made between the underlying framework: TensorFlow or Theano.

Theano is a Deep Learning library, implemented in Python, whose usage is quite extended in the data science community. One of its disadvantages is that the performance is low, as Python is interpreted, though it was not either a requirement.

Its usage consists on the implementation of a graph of tasks that it is lately compiled and executed. This makes it a very flexible framework, as the graph must be implemented from scratch. On the contrary, data structures in Theano are quite complicated, as if works at a very low level.

TensorFlow has some similarities with Theano, as it works by the implementation of a graph flow that is later compiled and executed. It is an open source software library for high performance numerical computation, though its usage by the community is highly related to Deep Learning.

As both frameworks are quite similar, TensorFlow was finally chosen to benefit from the author's previous expertise.

All python MIDI frameworks tend to convert MIDI bytes to an object oriented syntax. There are, however, some frameworks specifically designed to work with audio devices such as MIDI keyboards, DAWs, etc. This is not a required feature, as the framework is needed to extract MIDI note events and their attributes (pitch and time).

Two libraries stand out from the rest of them: Python-midi and Mido. python-midi is a framework originally designed to control audio equipment from a computer. It also allows to work with MIDI Messages, but it is not its main functionality. Mido is design to work with both MIDI Messages and Ports. Mido was finally chosen as the MIDI framework because of the extensive documentation it provides at the GitHub repository and its website.

3.2.3. Dataset

In the Magenta approach the dataset used was the published in the Classical Piano Midi Page. It contains 227 piano MIDI files from 25 composers and an additional Christmas section. The number of files was big enough but it was discarded as later experiments based on one single artist tend to have better results. This can be explained if we keep in mind that each piano composer has its own unique style and those features must also be learned by the network.

In the final design, four different datasets made by different communities were used to test the learning performance, but only two of them were finally selected for the evaluation process:

- Nottingham dataset: a set of over 1,000 folk songs originally written in ABC format and later converted to MIDI files. This is the biggest proven dataset, but the results were not satisfying, as the generation tended to get stuck in the most probable note.
- An Aerosmith dataset from FreeMidi.org containing 38 songs. The results were quite atonal so the dataset was discarded.
- A set of 16 songs from The Beatles extracted from Musescore.
- A set of 30 pieces of Bach extracted from Musescore.

MIDI files of each dataset were processed to extract the chords from the melody. A pre-processing algorithm was implemented to achieve that task. The following process was followed for each MIDI file in the data folder:

First, all MIDI files in the folder are shuffled so that generation occurs with different chords each time. For each MIDI file, it was converted to a JSON file and ticks per beat value was extracted (the note resolution, as explained in figure 2.6). Then, the script moved through each track of the MIDI file, and through each MIDI event inside that track.

For each MIDI 'Note on' event, a few note attributes were stored in two lists; harmony and melody:

- The played note.
- Note velocity.

- Related time to the past event.

For each MIDI 'Note off' event of 'Note on' event with 'velocity = 0', the stored attributes were:

- The played note.
- Related time to the past event.

Besides, the absolute time of the MIDI file was computed by the sum of all related times.

From each list, a one-hot encoding matrix was created where each Note on event was represented by 1.0 and each Note off event was represented by 0.0. The melody matrix had dimensions ['absolute_time', 36] and each row represented the active note in that tick. The chord matrix had dimensions ['absolute_time', 24] and a similar behavior

The note resolution of a MIDI file depends on its encoding. In particular, Musescore's sheet music has ticks per beat set to 480. That is a huge resolution that caused matrix dimensions to be more extensive than needed. Thus, a reduction of matrix dimensions was mandatory.

In order to reduce matrix dimensions, it was needed to set a note resolution. A sixteenth note seemed appropriate, as thirty-second notes are just common in percussion. A sixteenth note is represented with 4 ticks/beat so, for each row in both matrices, 1 of 120 values were taken (as $480/4 = 120$).

To better predict the note that could match a chord sequence, chords were sliced in sequences of 8 ticks, in order to predict a melody each two beats/quarter notes.

Final matrices have the following dimensions:

- Harmony matrix: [number of 2 beat sequences, 8 (ticks), 24 (2 octaves)]
- Melody matrix: [number of 2 beat sequences, 36 (3 octaves)]

3.2.4. Network architecture

Among all machine learning techniques, a Long Short Term Memory network was preferred since music needs to refer to structures that happened before posterior structures (e.g. the past chorus in a modern pop song).

The chosen data model determines two of the characteristics of the architecture: the input layer must have 24 LSTM and the output layer 36, in order to match input and output dimensions.

The batch-size for training is set to two mini-batches.

The test set size, which is the set used for generation is the 10% of the total of data instances. The validation set is the 30% of the total of data instances.

After the experimentations included in 4, a Mean Squared Error has been selected as loss function and the learning procedure consists on the Stochastic Gradient Descent.

The Dropout probability of non recurrent connections is set to 25%.

A threshold of 0.1 is set to transform the generated matrix of probabilities from the LSTM into a one-hot encoding matrix, in order to generate final MIDI file.

4. Experiments and Results

The section explains the experiments and analyses the results obtained on train, test and generation stages.

4.1. Preliminary experiments

At first, train and test losses through iterations did not strictly follow the usual negative exponential curve, as shows figure 4.1. In the figure, an irregular descending pattern with local minima in the validation set is shown.

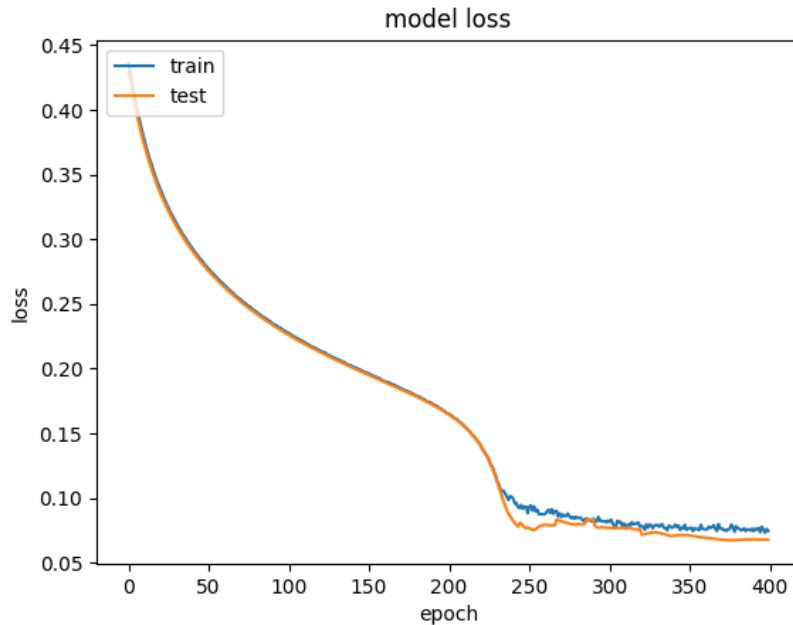


Figure 4.1: Train and test loss general curve profile

The image corresponds to one of the early experimentations made with The Beatles dataset. The parameters and loss results are shown in table 4.1:

In general, the most common issue detected in experimentation was the ability of the network to generate one or two most probable notes in an interleaving pattern.

This issue is not strange, because the network is learning to generalize the generation to the most repeated notes in the testing set. A high threshold also triggers this behavior.

Table 4.1: Experimentation parameters and results

Threshold: 0.02	Loss function: Binary cross-entropy
Number of layers: 2	Number of LSTM Cells per layer: 24,48
Validation set size: 20%	Test set size: 10%
Number of epochs: 400	Batches per epoch: 2
Dropout: 0.1	Recurrent dropout: 0
Train loss(@epoch = 400): 0.0747	Test loss(@epoch = 400): 0.0680

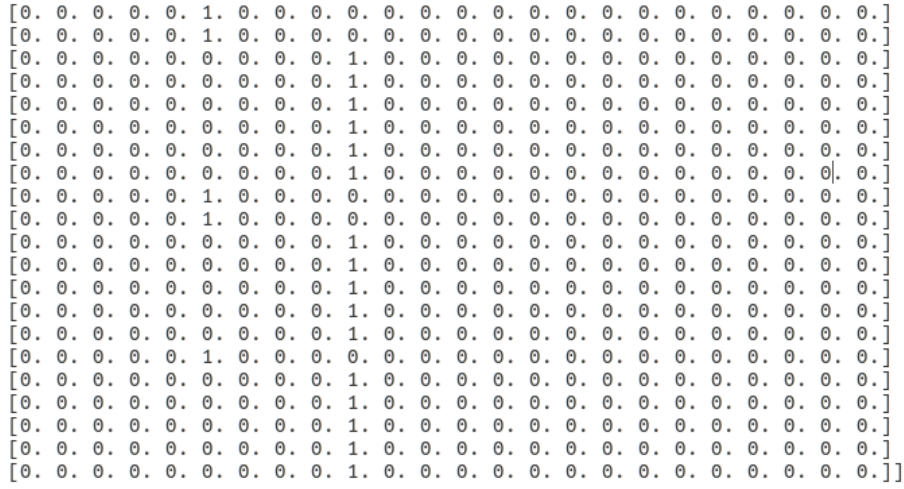


Figure 4.2: Generated melody matrix with two interleaved notes.

4.1.1. General observations

Over-fitting happens when the model is highly accurate, but only on the training set. It can be easily identified, as the loss curve in test differs heavily from the training curve.

In music generation, the same songs (or very similar ones) will always be generated in different experimentations because of the over-fitting. The model is learning to generate almost equal slices of the same notes of the training set each time.

Apart from that, another bad effect was noticed: all experimentations with hidden layers showed a better performance in training, as there were no irregularities in the loss profile, but generations stuck into one single note over and over. These experiments had 1 or 2 hidden layers with a number of LSTM cells multiple of 12 to provide a better performance with input and output layers.

Similar projects also describe this effect [31], even though two layers with limited LSTM cells (e.g. 9 and 18 cells in each layer) provide good results.

This note sticking problem led to the usage of simple input-output LSTM cells with no hidden layers in between.

4.2. Final music generations

Whenever tone note is more probable to be active than others, in a controlled way, a pleasant 'key effect' is detected, as this note uses to be the tonic of the other intervals. It is a good thing how they usually have an harmonic relation of an interval of 3rd or 5th (see figure 4.2, where D and G# are activated).

In any training procedure its expected for train and test to keep a similar curve, as the testing set must represent the training set. However, exceptions might happen when a dropout probability is applied to the neurons, as it only affects the training process.

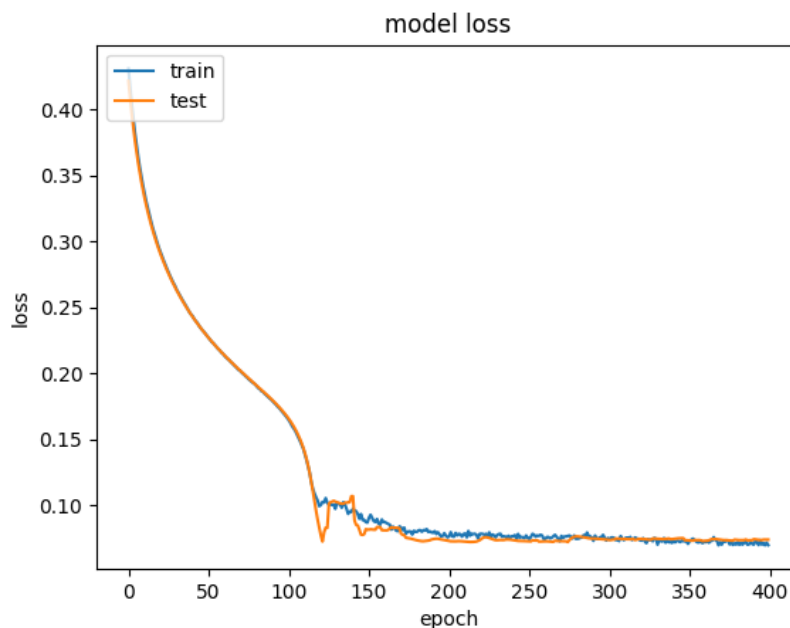


Figure 4.3: Train and test loss with Dropout: 0.25.

As some neurons are deactivated, the train loss takes a longer time than the test loss to be reduced and loss lapse might be unstable. Dropout had a significant effect on countering the generation to get stuck in the most probable note.

Stochastic gradient descent optimizer often works well with LSTMs [45] and it certainly helped with the irregular loss profile. By changing the loss function

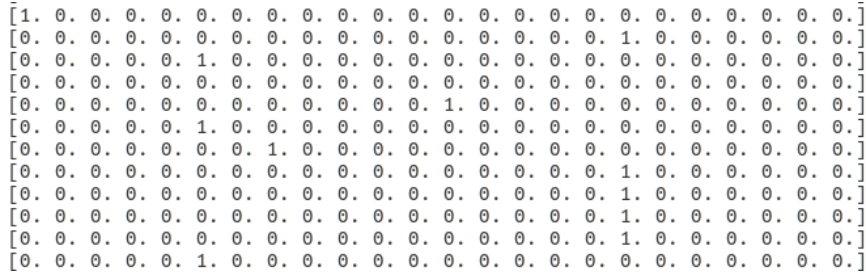


Figure 4.4: Generated melody matrix with dropout.

Table 4.2: Experimentation parameters and results with dropout

Threshold: 0.02	Loss function: Binary cross-entropy
Number of layers: 2	Number of LSTM Cells per layer: 24,48
Validation set size: 30%	Test set size: 15%
Number of epochs: 400	Batches per epoch: 2
Dropout: 0.25	Recurrent dropout: 0.1
Train loss(@epoch = 400): 0.0697	Test loss(@epoch = 400): 0.0739

from 'binary_crossentropy' to a classical 'mean_squared_error', and the optimizer from 'adam' to 'sgd', a much smaller loss is achieved in the first iteration, as well as a smooth negative exponential profile. The only disadvantage is that this combination is really slow; in our case, the loss curve follows an almost linear drop down (see figure 4.5) and takes many more epochs for the loss to converge.

After some experimentation it was concluded that good training and testing rates did not necessarily implied better generations, as it depended on the chosen dataset. Generation results with MSE and SGD are more dynamic, as it does not tend to get stuck in one or two notes. However, the generation often lacks of coherence, as dissonant intervals appear. The generations with MSE and SGD seems appropriate for Back-like generations but they sound dissonant in Beatles-like ones.

4.3. Online Listening Test

Music is meant to be listened by people. Therefore an online survey has been carried out to quantify if a music piece sounds pleasant to an audience and if it can be assigned to a particular genre. It was also interesting to know if the audience could differentiate a piece composed by a human versus a piece composed by an AI system.

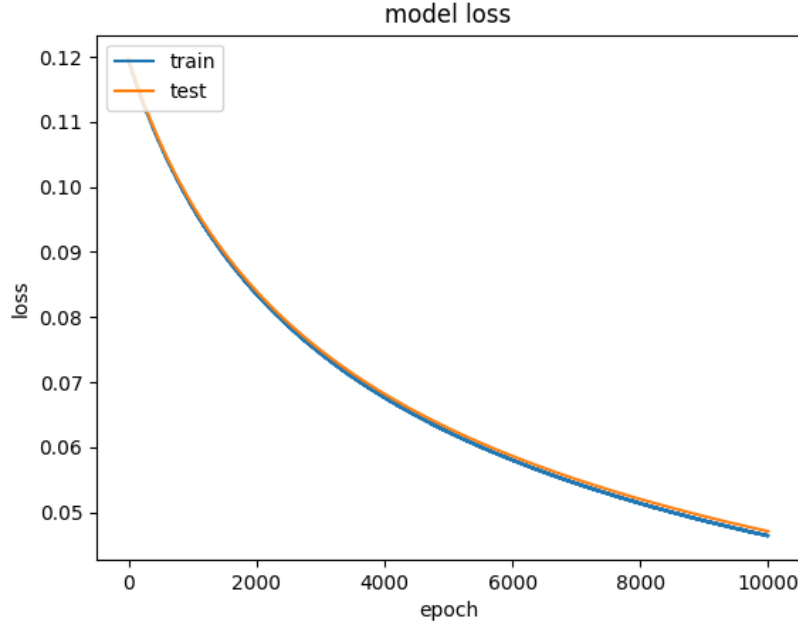


Figure 4.5: Loss when optimizer is changed to Stochastic gradient descent (SGD) and loss function to Mean Squared Error (MSE).

The survey is designed to ask three questions for each presented composition, alternating AI and human compositions. 120 human subjects from different ranges of age and musical background have participated. 2 compositions from The Beatles dataset and 2 from Bach dataset were shown (one composed from a human and the other one from AI in both cases).

Among all subjects, 67 (55.83%) had some kind of musical background (e.g. musical theory, piano, guitar,...), while 53 (44.17%) did not (see figure 4.6). This is perfect, as responses are quite balanced.

However, 87% of the subjects belong to a range from 18 to 25 years of age, as they are close people from college. Thus, a segmentation by age can not be properly done because of unbalanced classes.

4.3.1. Bach-like generation

With the Bach dataset, both AI and human generations cause doubts. While 'Nor pleasant neither unpleasant' has high rates in both cases, AI composition seems to be pleasant for a big amount of subjects.

Collected answers reveal that AI composition does not sound human-like

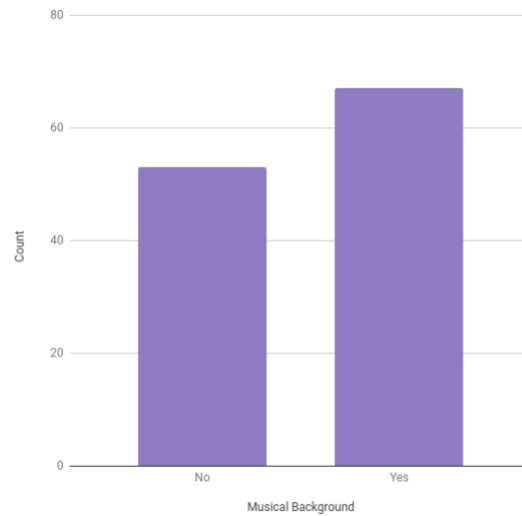


Figure 4.6: Count of subjects with and without musical background.

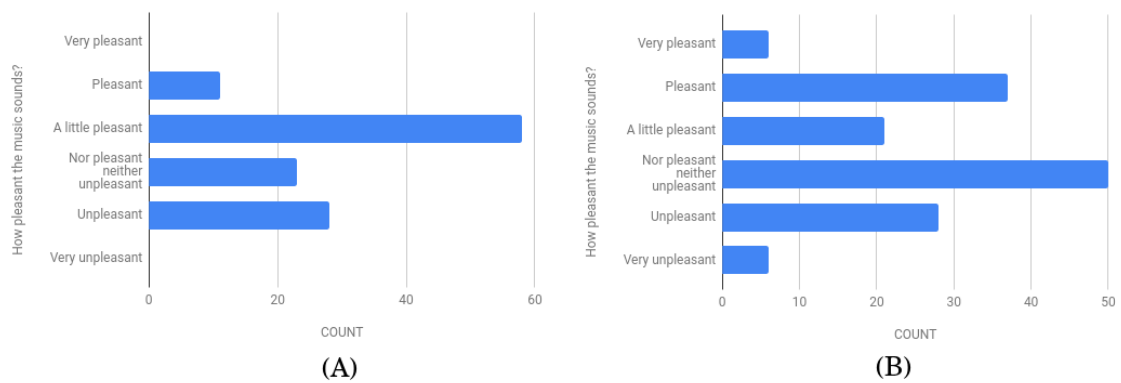


Figure 4.7: Responses to 'How pleasant the music sounds?' where (A) is a piece from a human composer and (B) from an AI for Bach's dataset.

for a slight majority of the subjects. However, this fact is not clear, as most of the subjects do not think the human composition are made by a human composer.

Table 4.3: Results for the comparative between Bach's dataset musical pieces.

Do you think the composition is made by a human?				
Human composition		%	A.I. composition	%
Yes	35	29%	63	53%
No	85	71%	57	48%

4.3.2. The Beatles-like generation

Beatles' generation seems to be more pleasant to hear. It is surprising that the A.I. composition sounds even more pleasing than the human composition. It could be said that the system generates a pleasant composition with Beatles data, which was the primary goal of the project.

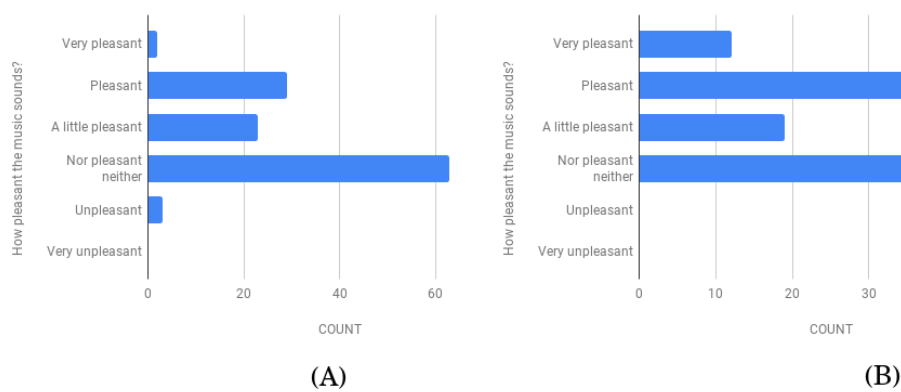


Figure 4.8: Responses to 'How pleasant the music sounds?' where (A) is a piece from a human composer and (B) from an AI for The Beatles' dataset.

In this case, it seems plausible that both compositions were composed either from an A.I. or a human.

Table 4.4: Results for the comparative between The Beatles' dataset musical pieces.

Do you think the composition is made by a human?				
Human composition		%	A.I. composition	%
Yes	54	45%	53	44%
No	66	55%	67	56%

5. Conclusions and further work

5.1. Conclusions

5.1.1. Accomplished or partially accomplished objectives

The primer goal of the project has been accomplished, as an autonomous system has composed a pleasant melody. In this case, the melody has been generated over a set of chords.

A functional system has been designed following common composition procedures to set simplifications that ease the representation problem, such as composing the melody over a set of chords and separating harmony notes from melody notes.

Common issues when generating music, such as stuck notes and local minima or over-fitting have been detected and solved.

The primer goal of the project has been accomplished, as an autonomous system has composed a pleasant melody. In this case, the melody has been generated over a set of chords.

5.1.2. Pending objectives

The composition based on The Beatles' dataset seems more attractive for people than the composition based on Bach's dataset. However, the test reveals that neither the pop, nor the classical music essence has been modeled according to the subjects.

How to measure a good computer composition is yet an unresolved problem, as usual metrics (e.g. log-likelihood) can not quantify how good a composition is.

There is not a clear conclusion about whether the subjects have been able to determine if the compositions were composed by a human or an AI or not.

5.2. Further work

Technical limitations of the survey platform have made impossible to answer the following questions:

- How many subjects with musical background think each one of the compositions are pleasant or unpleasant?
- How many subjects without musical background think each one of the compositions are pleasant or unpleasant?
- Is there a correlation between having musical background and better determining if the music piece is composed by a human or an A.I.?
- Is there a correlation between age range and thinking classical music is pleasant or unpleasant?
- Is there a correlation between age range and thinking pop music is pleasant or unpleasant?

Future versions of the project could include other datasets of jazz or blues performers. These two genres in particular have distinctive characteristics that could be easy to model.

There are lots of possibilities if a music genre is successfully modeled. A future line of work could answer the question if a determined genre is better modeled by instances of one artist or from various artists.

Finding paths to accurately extract chords from melodies in MIDI files would have a positive impact on final results.

Some RNN methods as teacher forcing could led to better models. It consists on feeding the output of a given time step to the input of the next time step.

The attention mechanism would perform a better modeling as an intuition of relevant information would be provided to the LSTM. It could also provide more freedom to the system, as a sequence to sequence architecture would then be mandatory. This way, input sequences could have different durations that output sequences.

Better modeling could led to a system that composes novel and pleasant melodies that appear to be composed by a human.

Metrics to quantify a good composition should also be defined. A good contribution could also define a method to use common metrics in a representative way.

5. References

- [1] V. Dignum, “Responsible artificial intelligence: Ethical thinking by and about ai.” [https://icps.gwu.edu/sites/g/files/zaxdzs1736/f/downloads/Virginia%20Dignum_%20Responsible%20Artificial%20Intelligence%20\(1\).pdf](https://icps.gwu.edu/sites/g/files/zaxdzs1736/f/downloads/Virginia%20Dignum_%20Responsible%20Artificial%20Intelligence%20(1).pdf).
- [2] C. y. D. Ministerio de Educación, “El registro de la propiedad intelectual.” <https://www.mecd.gob.es/cultura-mecd/areas-cultura/propiedadintelectual/la-propiedad-intelectual/preguntas-mas-frecuentes/registro.html>.
- [3] R. A. Española, “Autor, ra (query).” <http://dle.rae.es/?id=4UGeohY>.
- [4] J. Gibson, “Introduction to midi and computer music: The midi standard.” <http://www.indiana.edu/~emusic/361/midi.htm>.
- [5] T. M. Association, “<https://www.midi.org/specifications-old/item/about-smf-and-us-copyright-law>.” <https://www.midi.org/specifications-old/item/about-smf-and-us-copyright-law>.
- [6] J. S. Downie, “The music information retrieval evaluation exchange (2005–2007): A window into music information retrieval research,” *Acoustical Science and Technology*, vol. 29, no. 4, pp. 247–255, 2008. https://www.jstage.jst.go.jp/article/ast/29/4/29_4_247/_pdf/-char/ja.
- [7] W. Meurer, “Aspects of music information retrieval,” *School of Information at The University of Texas at Austin*. https://www.ischool.utexas.edu/~i385df04/StudentPPT/HTML/meurer_w/.
- [8] J. S. Downie, “Music information retrieval,” *Annual review of information science and technology*, no. 37, pp. 295–340, 2003.
- [9] R. B. Dannenberg, “Music representation issues, techniques, and systems,” *Computer Music Journal*, vol. 17, no. 3, pp. 20–30, 1993. <https://www.cs.cmu.edu/~rbd/papers/issues.pdf>.
- [10] M. Mathews and A. D. Nunzio, “Musien.” <http://www.musicainformatica.org/topics/music-n.php>.
- [11] “Csound: A sound and music computing system.”

- [12] “About midi-part 1:overview.” <https://www.midi.org/articles-old/about-midi-part-1-overview>.
- [13] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
- [14] A. Kurenkov, “A ‘brief’ history of neural nets and deep learning.” <http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/>.
- [15] R. Schapire, “Cos 511: Theoretical machine learning (lecture).” https://www.cs.princeton.edu/courses/archive/spr08/cos511/scribe_notes/0204.pdf.
- [16] J. Svegliato and S. Witty, “Deep jammer: A music generation model,” *Small*, vol. 6, p. 67.
- [17] J. M. Inesta, P. J. P. de León, J. Calvo-Zaragoza, and D. Rizo, “Genre-based melody generation through multi-objective genetic algorithms,” *MML 2015*, p. 16, 2015. https://www.researchgate.net/profile/Pauline_Mouawad2/publication/280490776_Multilabel_Classification_of_Non-Verbal_Communication_of_Emotions/links/5627882e08aee6327230b833.pdf#page=19.
- [18] A. Van Der Merwe and W. Schulze, “Music generation with markov models,” *IEEE MultiMedia*, vol. 18, no. 3, pp. 78–85, 2011.
- [19] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, “Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription,” *arXiv preprint arXiv:1206.6392*, 2012.
- [20] S.-C. Wang, “Artificial neural network,” in *Interdisciplinary computing in java programming*, pp. 81–100, Springer, 2003.
- [21] “Perceptron learning algorithm.” <http://lcn.epfl.ch/tutorial/english/perceptron/html/learning.html>.
- [22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [23] C. Olah, “Understanding lstm networks.” <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

- [24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
- [25] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
- [26] D. Eck and J. Schmidhuber, “A first look at music composition using lstm recurrent neural networks,” *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, vol. 103, 2002.
- [27] W. Zaremba, I. Sutskever, and O. Vinyals, “Recurrent neural network regularization,” *arXiv preprint arXiv:1409.2329*, 2014.
- [28] Magenta, “Models.” <https://github.com/tensorflow/magenta/tree/master/magenta/models>.
- [29] F. Liang, “Bachbot.” <https://github.com/feynmanliang/bachbot>.
- [30] F. A. Gers and J. Schmidhuber, “Recurrent nets that time and count,” in *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, vol. 3, pp. 189–194, IEEE, 2000.
- [31] K. Lackner, “Composing a melody with long-short term memory (lstm) recurrent neural networks.” http://konstilackner.github.io/LSTM-RNN-Melody-Composer-Website/Thesis_final01.pdf.
- [32] B. Krueger, “Classical piano midi page.” http://www.piano-midi.de/midi_files.htm.
- [33] W. Zhang, Z. Chen, F. Yin, and Q. Zhang, “Melody extraction from polyphonic music using particle filter and dynamic programming,” *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, vol. 26, no. 9, pp. 1620–1632, 2018.
- [34] ISO/IEC, *ISO/IEC 9899:TC3 - Programming languages — C*.
- [35] “News, status and discussion about standard c++.” <https://isocpp.org/>.
- [36] “Tensorflow website.” <https://www.tensorflow.org/>.
- [37] “Python website.” <https://www.python.org/>.
- [38] “Matlab website.” <https://es.mathworks.com/products/matlab.html>.

- [39] “Numpy website.” <http://www.numpy.org/>.
- [40] “Matplotlib website.” <https://matplotlib.org/>.
- [41] “Pandas website.” <https://pandas.pydata.org/>.
- [42] “Scikit-learn website.” <http://scikit-learn.org/stable/index.html>.
- [43] “Theano website.” <http://deeplearning.net/software/theano/>.
- [44] “Keras website.” <https://keras.io/>.
- [45] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, “Learning to learn by gradient descent by gradient descent,” in *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.